Software Development Kit for the System Visible Event Nexus Technology (SVEN)

User Guide

Document Number: 328506-001US
Revision: 1.0
World Wide Web: http://www.intel.com

Disclaimer and Legal Information

Contents

§

1 About this Document

This document describes the preview of the *Software Development Kit for the System Visible Event Nexus technology* (SVEN SDK). The SVEN SDK contains the source code for the event tracing infrastructure using the *System Visible Event Nexus technology* (SVEN) and related tools. You can use it to adapt the SVEN event tracing infrastructure to a Linux* platform. It also provides the necessary C/C++ include files and libraries for adding SVEN event instrumentation calls to kernel mode drivers and user space applications.

1.1 Intended Audience

The SVEN SDK is intended for software and validation engineers that plan to add SVEN software instrumented trace calls to their driver and application code. It provides the platform runtime infrastructure for collecting events and the development environment (headers and libraries) for instrumenting software modules.

1.2 Conventions and Symbols

The following conventions are used in this document.

Table 1 Conventions and Symbols used in this Document

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Indicates the exact characters you type as input. Also used to highlight the elements of a graphical user interface such as buttons and menu names.
This type style	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder.
[items]	Indicates that the items enclosed in brackets are optional.

{ item | item }	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.

§

2 Introduction

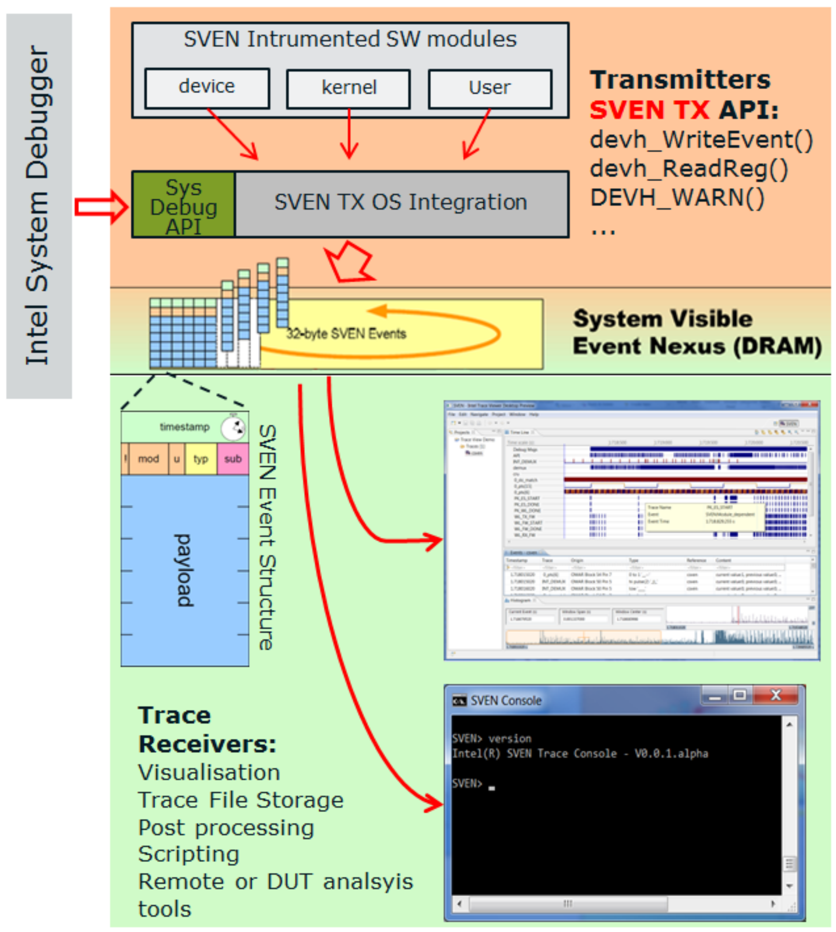
2.1 System Visible Event Nexus Technology (SVEN)

SVEN is a software technology (and API) that collects real-time, full-system visible software “event traces.” SVEN is currently built into Intel media/display drivers and is the primary debug tool for the Intel® Media SoC debug, performance measurement and regression.

SVEN is simply a list of software events with high resolution timestamps. The SVEN API provides developers a method of transmitting events from any operating system context and firmware.

2.2 SVEN Debug Infrastructure

The SVEN debug infrastructure consists of a small and fast "event transmit" (SVEN-TX) library and a verbose capture and analysis (SVEN-RX) capability for viewing and post processing. The TX library also contains debug hooks for the Intel system debugger. It enables the debugger to utilize SVEN instrumentation for advanced break conditions.



```
SVEN> version
Intel(R) SVEN Trace Console - V0.0.1.alpha

SVEN> _
```

3 Installing the SVEN SDK

3.1 Prerequisites

- Build the SVEN SDK on the same platform where you intend to run it. Installation using a cross-build environment like the Yocto Linux* pokey environment is not supported by this release.
- The SVEN SDK supports 32-bit and 64-bit Linux* systems using kernel 2.6.37 or higher. Building the SDK requires the GNU development tools for C/C++, GNU make and the kernel development environment for modules.
- The following command shows how to install the required components for a MeeGo 1.2 IVI release:

```
$ sudo zypper install kernel-adaptation-intel-automotive kernel-
adaptation-intel-automotive-devel make gcc gcc-c++
```

3.2 Installation

The SVEN SDK is provided in full source code. Follow the steps below to build and install it on your local system:

1. Go into the `sdk` folder of your local SVEN SDK copy.
2. Run `make` from inside the sdk folder.

   The SVEN SDK is built (see also Building the SDK).
3. Run `make install` with root privileges.

   The SVEN kernel modules, header files, tools and libraries are built (see also Building the SDK).
4. Verify the installation using the included `sventest.ko` driver and the csven debug console utility (see also Testing the SVEN SDK).

   Optional: Adapt the SVEN SDK to your needs and repeat the above build and install steps (see also Adapting the SVEN SDK).

4 Building the SVEN SDK

The SVEN SDK comes with a simple GNU make based build system. It allows building the SDK without changes and makes it easy to replace it with a custom build system used for the target platform.

The build system understands the following environment variables to customize the build:

Variable	Meaning	Default value
`SVEN_BSP`	Directory name of the SVEN board support package (BSP) inside the bsp folder that gets used during build	`example` The example BSP that ships with the SDK.
`KBUILDDIR`	Location of kernel module build environment	``/lib/modules/`uname –r`/modules`` The module build environment for the currently running kernel.

Use the following commands to build the SVEN SDK with default settings, which mean using the example SVEN SDK BSP and the currently installed kernel.

```
$ make clean
$ make
```

5 Installing the SVEN SDK File On the Local System

After building the Sven SDK successfully, run the following command to install the Sven SDK file on the local system. This step needs root rights as it copies the tools, kernel modules, libraries and header files into the system directories like `/usr/include`, `/usr/lib` and `/usr/bin`.

```
 $ sudo -E make install
```

6 Testing the SVEN SDK

This chapter assumes that the SDK was successfully build and installed.

Enter the following commands to test the SVEN SDK integration on your platform:

1. Load the sventest.ko driver that issues periodic SVEN events to the SVEN event nexus. This step requires root rights as it loads kernel modules into the running kernel.

   ```
   $ sudo modprobe sventest
   ```

2. Verify that both the sven and sventest kernel modules are loaded into the kernel using the following command:

   ```
   $ lsmod | grep sven
   sventest                1433  0
   sven                   25727  1 sventest
   ```

3. Check your system event log if the modprobe command above produced an error or if one of the two SVEN modules is not listed by lsmod. The event log is either shown on the console window or can be investigated with the following command:

   ```
   $ sudo tail  /var/log/messages
   ```

4. Run the SVEN console application csven:

   ```
   $ csven
   ```

   The console displays the output such as:

   ```
   SVEN Interactive: Built Mar 22 2012
   Copyright 2006-2012 Intel Corporation All Rights Reserved
   type help for a list of commands or quit
   (ev): 32 (long):4 (int):4 (short):2 (char):1
   SVEN-Header: ver:     'SVE2' disab: 00000000 debugfl:
   00000000
              : hdr_pa:35550000 hdr_sz:00001000
              : buf_pa:01c00000 buf_sz:00100000
   cb_count:00000001
   ```

```
SVEN-CBuf-0: cb_pa: 01c00000 cb_siz:00100000 cb_pos:
00000072 cb_id: 00000000
sven>
```

5. Enter the command “timestamp calc” to detect and validate the time stamp clock input from the BSP:

   ```
   sven> timestamp calc
   Calculating SVEN timestamp frequency for 30
   seconds.......... done

   Average sven clock ticks per second: 71211
   time:  1 wall clock: 01000202 sven: 00069925 delta -1286
   time:  2 wall clock: 02000439 sven: 00136149 delta -4987
   time:  3 wall clock: 03000594 sven: 00207586 delta +226
   …
   time: 28 wall clock: 28007436 sven: 01998913 delta +31454
   time: 29 wall clock: 29007660 sven: 02065132 delta -499

   Standard deviation of sven clock for 1 second durations :
   11179 (156981.123 us)
   WARNING: Clock source for sven is unstable !!!
   SVEN Timestamp Frequency (kHz):  71  0x00000047
   ```

   The command prints the red marked warning above if the timestamp clock produces unreliable results. Try adding the Linux* kernel boot option “idle=halt” if the CPU time stamp counter is used for calculation SVEN time stamps. This option disables Cx sleep mode utilization, which causes the TSC to stop counting on various CPU families.

   Verify that the frequency shown on the last line matches the value specified in the BSP.
   The default wall clock timer frequency is 1000 kHz.
   The frequency used by the TSC clock input is MAX_CPU_FREQUENCY / 1024, for example 1269 kHz on a 1.3 GHz CPU.

6. Enter the command “monitor” into the sven console. The console will display event sequences from the SVENTEST module.

   ```
   sven> monitor
   svenlog_thread_created
   dt:376716.584 mt:376716.584 M:00 U:00 T:debug_str S:log
   "SVEN:RBD_ABSENT"
   ```

```
dt: 388.965 mt:377105.549 API: call sventest_timer_tick(
pointer: 0x00000000.f84583b8 );
dt:   0.003 mt:   0.003 M:SVENTEST    U:00 T:debug_str
S:FuncEnter    "sventest_timer_tick"
dt:   0.002 mt:   0.002 M:SVENTEST    U:00 T:module_event
SLOW_TICK count:  83  jiffies: 03324252
dt:   0.004 mt:   0.004 M:SVENTEST    U:00 T:debug_str
S:FuncExit     "sventest_timer_tick"
dt:   0.001 mt:   0.001 API: retn sventest_timer_tick() =
0
dt:   9.986 mt:   9.986 M:SVENTEST    U:00 T:module_event
FAST_TICK count: 4142  jiffies: 03324262
…
```

7. To stop monitoring events, hit “Ctrl-C”. Then enter “quit” to exit the csven console.
8. Type the following command to unload the sven device drivers from your platform.

```
$ sudo rmmod sventest
$ sudo rmmod sven
```

7 Uninstalling the SVEN SDK

Run the following command while inside the top level SVEN SDK folder to uninstall the SVEN SDK. This step requires root rights at is removes the SDK distribution files from various system directories like `/usr/bin` and `/usr/lib`.

```
$ make uninstall
```

This step may produce several warnings and error messages which can safely be ignored.

8 Adapting the SVEN SDK

You can use the SVEN SDK as-is to get familiar with the SVEN infrastructure. Optionally, you can adapt the SDK to your needs in order to use its full potential.

To adapt the SVEN SDK to your needs, modify the “Board Support Package” (BSP) component of the SVEN SDK. The BSP contains the time stamp computation code and definitions for software modules that create events and definitions for custom event types. You can also add device register information to utilize the device register API portion of SVEN.

NOTE: Note: Alternatively, you can define modules and events using meta data files. This way you can use the original SVEN SDK and still be able to define your own modules and events. Use the `metaload` command to load the event decoding information dynamically into the SVEN console.

The following sections describe how to modify the BSP and define your own modules and event definitions for SVEN.

8.1 Create Your Own BSP

The first step in adapting the SVEN SDK is to create your own BSP based on the example BSP that ships with the SDK. Perform the following steps:

1. Copy the the BSP example folder to a name that describes your platform, for example `crownbay`.

   ```
   $ cd sdk/bsp
   $ cp –r example crownbay
   ```

2. Set the environment variable `SVEN_BSP` to match your BSP name. For example:

   ```
   $ export SVEN_BSP=crownbay
   ```

The build environment will now use the sources from the `bsp/crownbay` folder.

8.2 Defining SVEN Time Stamps

Each SVEN event is tagged with a 32bit time stamp that allows profiling and historically locating of SVEN events in a time line. The clock source used for the timestamp is defined in the following BSP file:

```
bsp/name/include/sven_timestamp.h
```

The symbol `sventimestamp()` returns the clock value as a 32-bit unsigned integer.
The symbol `sven_get_timestamp_frequency()` returns the clock frequency.

The example BSP ships with definitions for two different clock sources. It provides:

- Wall clock time using Linux* API functions with micro second resolution (1 MHz)
- CPU time stamp counter ticks at cpu_max_freq / 1024 resolution.

The wall clock time is used by default. This method works reliable but causes more overhead for transmitting an event. Uncomment the "`#define USE_TSC_TIMESTAMP`" in `sven_timestamp.h` to switch the BSP to use the time stamp counter as clock input.

NOTE: The timestamp counter is a very fast and low overhead access clock, but can be unreliable with sleep states. See also Adapting the SVEN SDK, which explains how to test the reliability of the used clock input.

8.3 Defining SVEN Modules

Each SVEN event is tagged with a module ID that defines its originating SW module. This module ID is used for various features like printing and filtering of events. Each code module instrumented with SVEN should be defined in the module table of the BSP. The files defining modules are

- `bsp/`*`your-bsp-name`*`/include/sven_module.h`
- `bsp/`*`your-bsp-name`*`/include/sven_mrd_table.h`

The file `sven_module.h` contains enumerations for the known modules and ID's for the module specific events that come from these modules.

The file `sven_mrd_table.h` contains the data structures for defining modules and its related module specific events. The file from the example BSP folder defines 2 modules called `VR_MyModule` and`SVENTEST`. These are meant for demonstration purposes and can be replaced with your own module definitions.

NOTE: The module `SVENTEST` is used by the sventest kernel mode driver stored under `driver/linux/test` of the SVEN SDK. You should keep the SVENTEST module in order to utilize this test driver for SVEN infrastructure validation (see also Testing the SVEN SDK).

To understand how the definition of SVEN modules work, search for the pattern `SVENTEST` and the module specific event definition `g_SVENTEST_specific_events` in the file `bsp/<bspname>/include/sven_mrd_table.h`. Then compare it with the usage in `driver/linux/test/sventest.c` that shows how the module definitions are used for code instrumentation. Search for the calls to `devh_SVEN_WriteModuleEvent` that issue module specific events defined by the BSP.

8.4 Defining API Events

SVEN offers predefined event types for API-Call tracing. These are typically used to instrument functions that are exposed to other SW modules or third-party code. The following SVEN instrumentation calls generate trace events for calling and returning from an API-call:

- `DEVH_API_CAL()`
  Create a trace event for entering an API call
- `DEV_API_RETURN()`
  Create a trace event for returning from an API call

An API call is identified by two numbers that are provided to the above SVEN instrumentation call. The first number defines an API function set. The second number defines the call inside the set. Both numbers are defined in the file `include/sven_api.h` of the used SVEN BSP. Pretty-printing of API events is done via the file `include/sven_api_table.h` of the used BSP. This file is used by the csven debug console to map the API-Id/Function-Id pairs from the instrumentation call to human readable information.

The example BSP contains one API call definition for demonstration purposes. The API is called `sven_test_timer_tick`. API instrumentation calls for this API are generated by the sventest example driver module in `driver/linux/test`. Search for `DEVH_API` in the file `driver/linux/test/sventest.c` to understand how API instrumentation calls are used.

9 Instrumenting User Mode Software

This chapter describes how to add SVEN instrumentation to a user mode software module. The following code shows a “hello world!” style SVEN instrumented application.

```
#include <stdio.h>
#include <sven_devh.h>

int main(int argc, char ** argv)
{
    /* get SVEN device handle */
    os_devhandle_t *devh = DEVH_FACTORY (NULL);

    /* assign module and unit ID */
    DEVH_SETMODULEUNIT (devh, SVEN_module_SVENTEST, 0);

    /* general purpose print event */
    DEVH_DEBUG(devh, "hello world!");
    /* cleanup handle */
   DEVH_DELETE (devh);
    return 0;
}
```

Use the following command line to compile the application:

```
$ gcc -o hello_sven hello_sven.c -lsven –lpthread
```

Run the csven debug console from a second terminal with the following command:

```
$ csven monitor
```

Run the “hello_sven” application. It produces the **bold** marked trace event output line in the csven debug console that originated from the `DEVH_DEBUG()` call in `hello_sven`.

```
$ csven monitor
svenlog_thread_created
 dt:   0.005 mt:336323.332 M:SVENTEST    U:00 T:debug_str S:log
"hello world!"
```

Inspect the file `sven_devh.h` and search for the pattern `DEVH_` to learn what other SVEN instrumentation calls exist.

10 Using the csven Debug Console

The csven utility is a console mode command for storing and displaying SVEN events directly on the platform. It can be uses both interactively and with scripts. To get an overview over the supported command, type `help`.

10.1 Quickstart: Capturing SVEN events into a Data File

Type the following commands to record all SVEN events into a data file. The bold font shows the commands, the normal font shows the tool output.

```
$ csven
SVEN Interactive: Built Mar 22 2012
Copyright 2006-2012 Intel Corporation All Rights Reserved
type help for a list of commands or quit
(ev): 32 (long):4 (int):4 (short):2 (char):1
SVEN-Header: ver:     'SVE2' disab: 00000000 debugfl: 00000000
           : hdr_pa:307fa000 hdr_sz:00001000
           : buf_pa:01c00000 buf_sz:00100000 cb_count:00000001
SVEN-CBuf-0: cb_pa: 01c00000 cb_siz:00100000 cb_pos:  003dab68
cb_id: 00000000
sven> hot enable all
hdr disable: 0x00000000 -> 0x00000000
sven> stream events.bbr
svenlog_thread_created
```

Now execute the workload that issues the events to capture. Type Ctrl-C when you are done.

```
sven> stream events.bbr
svenlog_thread_created
^CUSER-BREAK, exiting monitor back to console
wrote 1395 events to file "events.bbr"
sven>
```

The csven utility stops saving the events and reports how many events where written into the given file.

10.2 Loading Events From a File

You can use the load and dump commands to load and view a previously recorded event file.

```
sven> load events.bbr
eading events from file "xx.bbr"
read 1395 of 1395 events from file "xx.bbr" filtered
Event Capture is now paused, you must type run to capture again
sven> dump
dt:-391.000 mt:-391.000 M:SVENTEST    U:00 T:module_event
FAST_TICK count: 3693516  jiffies: 76588898
 dt:  10.003 mt:  10.003 M:SVENTEST    U:00 T:module_event
FAST_TICK count: 3693517  jiffies: 76588908
 dt:   9.992 mt:   9.992 M:SVENTEST    U:00 T:module_event
FAST_TICK count: 3693518  jiffies: 76588918
 dt:  10.302 mt:  10.302 M:SVENTEST    U:00 T:module_event
FAST_TICK count: 3693519  jiffies: 76588928
 …
```

10.3 Event filtering

There are two types of event filters. This first one is a global event mask called the hot enable mask. The second one is a display filter inside the csven debug console. It controls which events are loaded into the csven local memory buffer and get shown using the dump and monitor commands.

10.3.1 HOT filtering

The hot command is used to enable/disable SVEN Event transmission into the nexus. Events that are hot disabled will not be seen by any logging application. This is the most convenient way to turn off debug instrumentation transmission into the nexus, even while streaming software is executing.

The command syntax is:

`hot enable` *`category`* - enable system-wide transmission of an event category into the nexus
`hot disable` *`category`* – disable system-wide transmission of an event category into the nexus

Where *`category`* is one of:

- `all` - All Events

- `strings` - General debug strings transmitted by calls to
  `DEVH_DEBUG( devh, str )`
  `DEVH_WARN( devh, str )`
  `DEVH_FATAL_ERROR( devh, str )`
- `regio` - All hardware register reads/writes transmitted by calls to
  `devh_ReadReg32( devh, offset )`
  `devh_WriteReg32( devh, offset, value )`
- `func` - driver execution sequence events, transmitted by
  `DEVH_FUNC_ENTERED( devh )`
  `DEVH_FUNC_EXITED( devh )`
  `DEVH_AUTO_TRACE( devh )`
- `mod`- Module-specific events
- `api` - SVEN API call/return trasmitted by
  `DEVH_API_CALL(devh, api, func, p0, p1, p2, p3, p4)`
  `DEVH_API_RETURN(devh, api, func, rt)`

When you use `filter reject all`, use also `filter accept` *`event-specification`* to re-enable the filter for the specified kind of events. See also Display filtering.

10.3.2 Display filtering

The SVEN Debug console, by default, records *all* events being transmitted into the nexus. The filter command is used to selectively enable or disable recording of these events into the debug console's local memory (for display by the dump or monitor commands)

The command syntax is:

```
sven> filter help
ERR: usage filter [accept|reject] event-specification
```

Where:

- `filter accept` *`event-specification`*
  Allows *`event-specification`* to be recorded locally.
- `filter reject event-`*`specification`*
  Rejects *`event-specification`*, do not record locally.
- `filter reject all`
  Sets default to not record events locally.
- `filter accept all`
  Sets default to record all events locally.

More complex event-specifications are given in tuples, for example:

`filter reject module SVEN_TEST`
Do not record any event written by module SVEN_TEST
`filter reject event register_io`
Do not record register IO Events transmitted by any device.

`filter reject module SVEN_TEST event module_event`
Do not record module specific Events transmitted by `SVEN_TEST`.

To find options available for a tuple, submit a question mark in its place to get a list of available options:

```
sven> filter accept event ?
event types [invalid trigger debug_str register_io
             port_io module_isr os_isr os_thread
             smd module_event api]sven>

filter reject event debug_str subtype ?
subtypes: [invalid log FuncEnter FuncExit AutoTrace
           FuncInvalidParam Checkpoint Assert Warning
           FatalError PresTiming ]
```

10.4 Offline Viewing of Events from a Binary Event File

This is an example script to extract and print API events from a binary SVEN event log file. Save this test into a file called sift.sven.

```
#sift.sven
filter reject all
filter accept event api
load event.bbr  -- Binary sven log
time tminus
dump 1000000  -- Number to dump(large number extracts all events)
```

Type the following command to run the script:

```
$ csven source sift.sven
```

It will produce output such as:

```
Sourcing script "dump.svn"
SVEN:
SVEN: filter reject all
SVENLog default rejects all events
SVEN: filter accept event api
Adding Filter:
 EV: 00000000 000c0000 00000000 00000000 00000000 00000000
00000000 00000000
 MSK:00000000 003f0000 00000000 00000000 00000000 00000000
00000000 00000000
SVEN: load event.bbr  ------ Binary sven log
reading events from file "event.bbr"
read 50 of 1395 events from file "event.bbr" filtered
Event Capture is now paused, you must type run to capture again
SVEN: time tminus
time display mode is now 2
SVEN: dump 1000000  -------- Number of events to extract (large
number extracts all events)
 t-:24048.023 API: call sventest_timer_tick( pointer:
0x00000000.f84583b8 );
 t-:24048.009 API: retn sventest_timer_tick() = 0
 t-:23046.055 API: call sventest_timer_tick( pointer:
0x00000000.f84583b8 );
 t-:23046.043 API: retn sventest_timer_tick() = 0
 t-:22044.038 API: call sventest_timer_tick( pointer:
0x00000000.f84583b8 );
 t-:22044.024 API: retn sventest_timer_tick() = 0
 t-:21042.050 API: call sventest_timer_tick( pointer:
```

```
0x00000000.f84583b8 );
…
```

10.5 Handling SVEN Time Stamps

The `timestamp` command can be used to show, set and auto-detect the SVEN time stamp frequency used for computing event time information in csven.
Type `timestamp ?` to get an overview about the variants of the timestamp command.

Enter the following command to show the current timestamp frequency used by csven :

```
sven> timestamp
SVEN Timestamp Frequency (kHz):  1269  0x000004f5
```

Enter the following command to change the timestamp frequency value. You can use `khz`, `mhz` or `ghz` suffixes to specify the magnitude of the frequency:

```
sven> timestamp 1mhz
timestamp frequency set to 1000 kHz.
```

Enter the following command to auto-detect the used timestamp frequency for SVEN event based on 30 one second intervals. The command also computes the standard deviation of the clock from its 1 second tick average and gives a warning if the clock is unstable (see also Defining SVEN Time Stamps).

```
sven> timestamp calc
Calculating SVEN timestamp frequency for 30 seconds
.............................. done

Average sven clock ticks per second: 1000141

time:  1 wall clock: 01000169 sven: 01000168 delta  +27
time:  2 wall clock: 02000298 sven: 02000298 delta  -11
time:  3 wall clock: 03000434 sven: 03000434 delta   -5
time:  4 wall clock: 04000564 sven: 04000563 delta  -12
…
time: 28 wall clock: 28003982 sven: 28003982 delta   -5
time: 29 wall clock: 29004112 sven: 29004112 delta  -11

Standard deviation of sven clock for 1 second durations : 19
(18.955 us)

SVEN Timestamp Frequency (kHz):  1000  0x000003e8
```

11 Command Reference

This chapter describes the csven console command set. Type “help” to print the list of available console commands.

11.1 quit

Exit the SVEN console.

11.2 help

Print list of available commands, including a short description of its purpose.

11.3 sleep

Syntax: `sleep number`

Suspend command processing for the specified number of seconds.

11.4 pause

Immediately stop capturing additional events into local memory.

NOTE: This does not prevent events from being transmitted into the Nexus. It just prevents the console from capturing them. In this mode, you can inspect recently transmitted events, usually using the dump command.

11.5 run

Immediately resume capturing additional events into local memory. This is often used to resume capture after a trigger fires.

11.6 monitor

Print all events that are coming into the nexus. This command keeps running until a trigger fires or `Ctrl-C` is pressed.

11.7 hdr

Print SVEN shared memory header information.

11.8 dump

Syntax: `dump [number]`

Show the recently received events in the nexus. The parameter *`number`* defines the maximum number of events shown. The default for *`number`* is 40.

11.9 hexdump

Show the recently received events in the nexus in raw hex format. The parameter *`number`* defines the maximum number of events shown. The default for *`number`* is 40.

11.10 search

Search events for any payload between *`min`* and *`max`*.

11.11 lookup

Lookup a module, register, or bitfield.

11.12 modules

List supported modules from BSP or meta data.

11.13 logwrite

Create various events from the console into the nexus. This is used to demonstrate various instrumentation API calls and to test event transmission into the SVEN nexus.

11.14 source

Execute commands from a csven script file.

11.15 thread

Launch svenlog monitor thread to collect events from the nexus.

11.16 decode

Decode `MODULE` *`reg_offset reg_value`* `[`*`prev_value`*`]` to text (requires register data definitions in the BSP).

11.17 peek

Read a named register
(requires register data definitions in the BSP).

11.18 poke

Write a named register
(requires register data definitions in the BSP).

11.19 hot

Hot enable/disable of event writer categories.

The hot gate is a global filter. It controls which event types that will be transferred into the event nexus. Using "hot disable all" turns off any SVEN logging on a system.

Syntax: `hot [enable|disable] [all|strings|regio|func|smd|mod|perf|api|fw]`

11.20 trigger

Trigger on a specific event. Event capture will stop when the trigger matching even is seen on input. See also: triggerdelay and triggerwait.

11.21 triggerdelay

Set the number of events to record in addition after a trigger fires.

11.22 triggerwait

Wait for a trigger event [optional timeout in seconds].

11.23 filter

Filter SVENLOG based on a mask provided (see also section "Event filtering").

11.24 save

Save the binary SVENLOG to a file.

11.25 load

Load a binary SVENLOG from a file. The console stops monitoring the nexus and uses the provided file as the trace input buffer.

11.26 metasave

Save bsp metadata into a file. This metadata contains all BSP definitions like modules, module events and APIs.

11.27 metaload

Load bsp metadata from a file. The metadata replaces the BSP information and can be used to provide event pretty printing information without the need to compile an own BSP.

Syntax: `metaload [-mv] filename`

The option `–m` merges the metadata with the current information instead of replacing it. This option is used to add BSP definition incrementally to the console.

The option `-v` provides verbose output on the action taken by metaload. It is useful for analyzing problems when using manual modified metadata files.

11.28 metashow

Print currently used metadata information.

11.29 metareset

Revert to the “compiled-in” BSP metadata.

11.30 time

Select time display mode.

11.31 overhead

Measures SVEN instrumentation overhead.

11.32 stream

Stream the SVENLOG to a file. Events are saved until a trigger fires or `Ctrl-C` is pressed.

11.33 timestamp

Set or compute timestamp frequency.

Syntax:

`timestamp` *`num`*`[GHz|MHz|KHz`	set timestamp frequency
`timestamp calc`	compute timestamp frequency
`timestamp`	show timestamp frequency

11.34 systime

Print the current system time.

11.35 echo

Print a string.

12 Understanding the SVEN SDK Folder Structure

The SVEN SDK is delivered as a source tree. The following table explains the contents of the various folders in the distribution:

Folder Name	Description
`bsp`	Root folder of the SVEN board support package adaptation tree .
`bsp/example`	Example BSP that ships with the SVEN SDK. Adaptations to the SVEN SDK are usually done by copying this folder to a name matching the targeted platform.
`bsp/example/include`	Definitions for SVEN modules, SVEN module events, SVEN API events and device registers.
`driver/linux`	Location of the SVEN device driver source code for Linux. This driver provides the infrastructure for storing SVEN event in a system and user mode visible memory buffer. It also exposes the SVEN instrumentation API to kernel modules.
`driver/linux/test`	A “hello world” style device driver that shows how to use instrumentation calls. It uses the following event types: Module specific events from the example bsp (`devh_SVEN_WriteModuleEvent`) API events from the example bsp (`DEVH_API_CALL`, `DEVH_API_RETURN`) Generic Func enter/exit events (`DEVH_FUNC_ENTER`, `DEVH_FUNC_EXIT`) Generic output events (`DEVH_DEBUG`)
`include`	Public include files for the SVEN infrastructure
`osal`	Location of an OS abstraction layer. This folder increases the portability of the generic SVEN code by providing a generic API to OS specific primitives

	(memory mapping, thread synchronization …). This code is usually used “as-is” without the need for modifications.
`src`	Shared source code of the SVEN infrastructure. It is both used by the kernel mode driver and the user mode libraries.
`tools/csven`	The source code of the SVEN debug console. This code is used to configure the SVEN runtime and to visualize or save trace data.
`tools/dumpbbr`	A simple utility to dump the raw SVEN event data saved by the debug console. It can be used as a starting point for writing an own event data post processing application.